3 Printer Problems

(25 Points)

It's midterm season, and the Soda printers are working overtime to print out everyone's files. Help the printers print everything out in the right order!

We've defined a `PrintJob` class below.

```
public class PrintJob {
    List<String> pages; // a List of Strings representing the pages of the printout
    int numCopies; // an int denoting the number of copies to make.
    public PrintJob(List<String> pages, int copies) {
        assert pages.size() > 0 && copies > 0; // pages.size() and copies will be greater than 0.
        this.pages = pages;
        this.numCopies = copies;
    }
}
```

A `Printer` can be modeled as an iterator, behaving as follows:

- **public** `Printer()`: Creates a new printer with no print jobs. Contains a `jobs` deque.
- **public void** `sendJob(PrintJob job)`: Sends a print job to the back of the `jobs` deque.
- **public** `String next()`: Returns the next page to be printed. The printer should print a page from the first job in the `jobs` deque. When the job is completed, the job should be removed from the `jobs` deque, and the printer should begin printing the next PrintJob.
  - In order to print a job, exactly `numCopies` of the strings in `pages` should be printed, with the pages collated. For example, let's say we had a PrintJob with `pages = List.of("1", "2", "3")` and `numCopies = 4`. We expect `Printer` to output in the following order: `123123123123` (instead of `111122223333`)
- **public boolean** `hasNext()`: Returns if the printer has a next page to print.

For example, if we ran the following program:

```
Printer p = new Printer();
p.sendJob(new PrintJob(List.of("N", "a"),13));
p.sendJob(new PrintJob(List.of("Bat", "ma", "n"), 2));
for (int i = 0; i < 3; i++) {
    System.out.println(p.next());
}
p.sendJob(new PrintJob(List.of("!"), 5));
while (p.hasNext()) {
    System.out.print(p.next());
}
```

We should get the following output:

```
N
a
N
aNaNaNaNaNaNaNaNaNaNaBatmanBatman!!!!!
```

(a) Implement the `Printer` class methods.

```
public class Printer implements Iterator<String> {
    public Deque<PrintJob> jobs;
    public int currPage;
    public int currCopy;

    public Printer() {
        this.jobs = new ArrayDeque<PrintJob>();
        _____1_____;
        _____2_____;
    }

    public void sendJob(PrintJob job) {
        _____3_____;
    }

    @Override
    public boolean hasNext() {
        return !this.jobs.isEmpty();
    }

    @Override
    public String next() {
        if (_____4_____) {
            throw new NoSuchElementException("No more pages to print.");
        }
        PrintJob currJob = _____5_____;
        String nextPage = _____6_____;
        this.currPage = _____7_____;
        if (currPage == currJob.pages.size()) {
            _____8_____;
            _____9_____;
        }
        if (currCopy == currJob.numCopies) {
            _____10_____;
            _____11_____;
            _____12_____;
        }
        return nextPage;
    }
}
```

(b) Halfway through printing the midterm, you realize that there's a mistake! Fortunately, you kept a reference to the PrintJob `pj` you sent, so you update the PrintJob. For each of the following updates, **select all** options that could happen.

You may assume that the PrintJob had started, but not completed when you updated the PrintJob (at least one page of the Job has been printed, and at least one page of the Job has yet to be printed). The PrintJob gets modified between `next()` and `hasNext()` calls (i.e. not during `next()` or `hasNext()` calls). After the update, you call `next()` until `hasNext()` `returns` **`false`**.

i. You add finitely many additional pages to the end of the exam (ex. with `pj.pages.addLast();`)
☐ The PrintJob finishes as if the job hadn't been updated
☐ The PrintJob finishes as if the job had been updated from the start
☐ The PrintJob finishes, but some copies look like the old version, and some copies look like the new version
☐ The PrintJob never finishes
☐ The Printer crashes
○ None of the above

ii. You remove some (but not all) pages from the end of the exam (ex. with `pj.pages.removeLast();`)
☐ The PrintJob finishes as if the job hadn't been updated
☐ The PrintJob finishes as if the job had been updated from the start
☐ The PrintJob finishes, but some copies look like the old version, and some copies look like the new version
☐ The PrintJob never finishes
☐ The Printer crashes
○ None of the above

iii. You increase the number of copies (ex. with `pj.numCopies += 100;`)
☐ The PrintJob finishes as if the job hadn't been updated
☐ The PrintJob finishes as if the job had been updated from the start
☐ The PrintJob never finishes
☐ The Printer crashes
○ None of the above

iv. You decrease the number of copies (ex. with `pj.numCopies -= 100;`) After this change, `pj.numCopies` is still positive.
☐ The PrintJob finishes as if the job hadn't been updated
☐ The PrintJob finishes as if the job had been updated from the start
☐ The PrintJob never finishes
☐ The Printer crashes
○ None of the above